Approaches for Preventing Honeypot Detection and Compromise

Michail Tsikerdekis
Computer Science Department
Western Washington University
Bellingham, WA, USA
Email: Michael.Tsikerdekis@wwu.edu

Sherali Zeadally
College of Communication and Information
University of Kentucky
Lexington, KY, USA
Email: szeadally@uky.edu

Amy Schlesener
Computer Science Department
Western Washington University
Bellingham, WA, USA
Email: schlesa@wwu.edu

Nicolas Sklavos
Computer Engineering & Informatics Department
University of Patras
Patra, Greece
Email: nsklavos@ceid.upatras.gr

***Abstract*—Honeypots have been used extensively for over two decades. However, their development is rarely accompanied with an understanding of how attackers are able to detect them. Further, our understanding of effective evasion strategies that prevent the detection of honeypots is limited. We present a classification of honeypot characteristics as well as honeypot detection evasion strategies which minimize the detection rates of honeypots. We also provide recommendations for future honeypot software which is more adaptable, modular and incorporate a dynamic intelligence design.**

***Index Terms*—Detection, honeypot, protocol, security**

I. Introduction

Honeypots have been used as a standard mechanism to provide a more holistic cybersecurity strategy to organizations for over two decades [1]. They have been credited with directly assisting the discovery of a system's compromise and in some cases, they have also helped to uncover the identity of cyber-criminals across countries [2]. However, the proliferation of tools and technologies such as machine learning and artificial intelligence has enabled attackers to not only detect honeypots but identify ways for further exploitation. To date, much of the literature has focused on the development of novel honeypots that often target a particular aspect (such as defending a Demilitarized Zone (DMZ) zone) without providing a holistic understanding of how honeypot can be designed so as to avoid detection by attackers.

Today, all types of organizations face probing from a globally connected cyberspace. A recent study of five small public municipalities from the state of Washington has demonstrated that even smaller networks are not immune to probing by a range of attackers from various countries [3]. Organizations employ honeypots to some degree as a mechanism for detecting system probing and compromise from both internal and external threats. Since the intent behind honeypots is for them to be exploited by attackers, any traffic originating from a honeypot is suspect [4]. Specifically, the rate of false positives by a honeypot is expected to be virtually zero and as such the trust placed in alerts coming from such systems by cybersecurity analysts is high. Honeypot detection by an attacker often boils down to a comparison of fidelity (how realistic a honeypot's behavior is compared to the system it mimics). However, studies have shown that high fidelity honeypots can still be detected and are expensive to maintain [5], [6].

Contributions of this work

We summarize the main contributions of this work as follows:

- We identify the key characteristics that differentiate various honeypots.
- We review some of the recent approaches that have been found to make honeypots more difficult to detect by attackers and classify which honeypots benefit the most from these approaches.
- We discuss some of the characteristics that future honeypots need to have in order to keep up with emerging threats.

The rest of the paper is organized as follows. In section II, we present an overview of various honeypots and their respective uses along with their key characteristics. In section III, we identify how honeypot characteristics influence their ability to avoid detection by attackers and provide a classification for such characteristics for developers of honeypots. Finally, in section IV, we present a few characteristics that future honeypots need to have with respect to the tools that are available to attackers.

II. Honeypots

At its simplest, a honeypot is a resource that is utilized to detect and prevent attempts of unauthorized access and use of systems. Its value is found in its exploitative state [1]. The expectation is that an attacker will interact with the honeypot and will unknowingly provide information to cybersecurity professionals revealing his or her whereabouts and intent. The latter is of utmost importance one of the most important defense measures is to quickly detect zero day vulnerabilities (vulnerabilities not previously known) and reacting to such compromises. As such, the information gained through this process can be used to uncover such vulnerabilities or help minimize risks associated with an attacker's specific attack goals, which lead to attack vector customization that may be difficult to otherwise detect. Offensive honeypots include honeypots involving drive-by-downloads [7] and P2P honeypot bots developed to control and monitor botnets [8]. Honeypots are also used in network areas that cannot be otherwise effectively monitored by network monitoring tools. For example, once an employee's network access credentials are obtained by a remote attacker (e.g., Secure Shell (SSH) credentials), intrusion detection systems that are often placed near the edge router of a network are unable to inform security analysts about the actions of the attacker. A honeypot, however, can indicate the compromise of the employee's account if the attacker attempts to connect to it via the employee's workstation. An example of this is presented on Fig. 1.

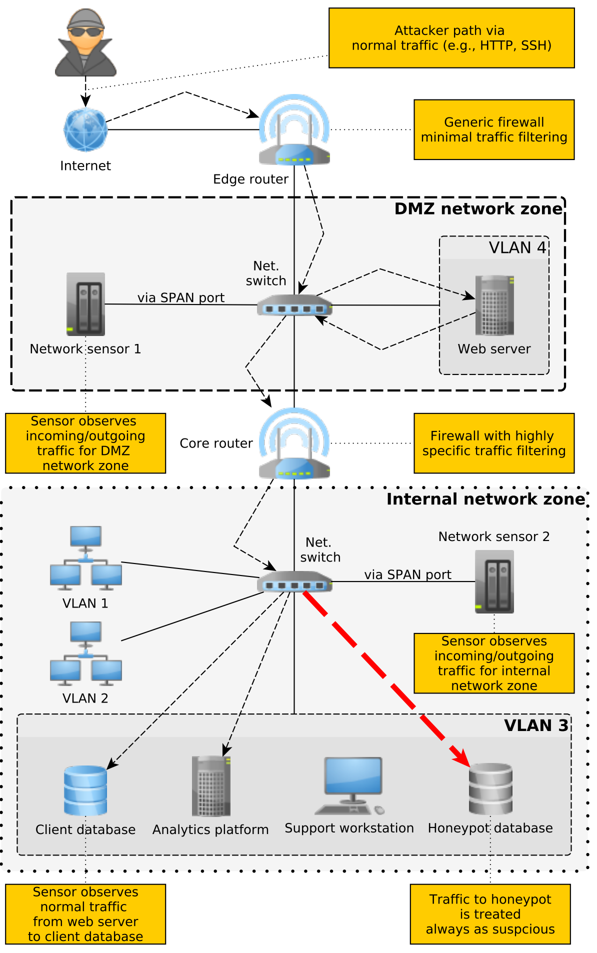
Attacker path via normal traffic (e.g., HTTP, SSH)
Generic firewall minimal traffic filtering
Internet
Edge router
DMZ network zone
VLAN 4
Net. switch
via SPAN port
Network sensor 1
Web server
Sensor observes incoming/outgoing traffic for DMZ network zone
Core router
Firewall with highly specific traffic filtering
Internal network zone
Net. switch
Network sensor 2
via SPAN port
VLAN 1
VLAN 2
Sensor observes incoming/outgoing traffic for internal network zone
VLAN 3
Client database
Analytics platform
Support workstation
Honeypot database
Sensor observes normal traffic from web server to client database
Traffic to honeypot is treated always as suspcious

Fig. 1. Example attack scenario on a network with network intrusion detection sensors and honeypot.

Honeypots today are extensively used in a range of governmental as well as non-governmental organizations. For example, one of the requirements by the Health Insurance Portability and Accountability Act (HIPPA) that governs healthcare security requirements in the United States is interpreted as a need for monitoring systems. Often this requirement for healthcare providers as well as business associates interpret this as a need for Intrusion Detection Systems (IDS) that include the use of honeypots to some extent [9]. Other critical areas that make use of honeypots include financial institutions [10] and industrial control networks [11].

The type of data collected by honeypots vary depending on the organization and the type of application but in general they involve extensive logs of user and system actions as well as network traffic. These logs are often forwarded in a larger data storage (e.g., Elasticsearch) that contains network sensor data (e.g., netflow or full packet captures) along with alerts from IDS. Alert correlation is further used in order to better establish a timeline for an attacker's actions or eliminate false positives [12]. Security analysts then analyze the honeypot data. However, unlike other network data (full packet captures, netflows, and so on) that is collected, the honeypot data is more valuable and its analysis is a lot cheaper because it contains less false positives [13].

A. *Honeypot Characteristics*

Various types of honeypots have been developed for a diverse set of applications. Some are more generic whereas some are highly specific characteristics such as the ability to emulate a particular network encryption (e.g., Secure Sockets Layer Version 3 (SSLv3)). We have identified several honeypot characteristics and we have also reviewed their major benefits as well as their drawbacks. Next, we discuss the following honeypot characteristics: objective, fidelity, implementation and scalability.

1) Objective: There are two primary categories for honeypots: *research* and *production* [14]. Research honeypots are used to generate threat intelligence about attackers. They are often placed in a network's DeMilitarized Zone (DMZ) or otherwise in an area where production activities do not take place (e.g., through Virtual Local Area Networks (VLAN) segmentation). In contrast, production honeypots are often placed near security assets with the purpose of serving as indicators of compromise(IOC) for internal as well as external threats. Simply put, the functionality of these honeypots are for alerting defenders of an attack, and only limited intelligence is gathered about the attacker (in sharp contrast with research honeypots). The earliest precursor of production honeypots is monitoring darknets (the unused IP address space in a local network) [13]. The general principle of production honeypots is that since they serve no functional purpose for legitimate users, any attempt to access and use them is considered to carry a bad intent [15]. From an attack perspective, the difference between production and research can be thought in terms of "attack duration" with research honeypot involving longer times.

2) Fidelity: Honeypots can be characterized based on the level of interaction that they allow between them and the attacker. *Low interaction* honeypots allow for interactions with them only for a short period of time. At the completion of the interaction an attacker may or may not discover the deceptive nature of the honeypot depending on the intent of the honeypot developers. This may be the case with an OpenSSH honeypot server that follows the SSH protocol and accepts various authorization credentials but provides no further shell functionality (revealing in effect to the attacker the deceptive nature of the honeypot). Such a honeypot is typically placed in a local network's DMZ or within an IP subnet whose network traffic is not monitored by an IDS. On the other hand, *high interaction* honeypots can either simulate or provide all high level functions of a system they are mimicking. The expectation here is that their deceptive nature should not be detectable by an attacker. Often such honeypots are used exclusively for research where the main objective is to discover the intent and methods of an attacker [14]. High interaction honeypots are usually used where zero day vulnerabilities are more likely to be discovered [1].

3) Implementation: A honeypot implementation involves the use of *software*, *hardware* or *hybrid*. Hardware honeypots can vary from regular computers to specialized Supervisory Control and Data Acquisition (SCADA) systems (e.g., GasPot, a gas pump SCADA honeypot). The degree of emulation of functions based on the system that a honeypot mimics can vary but often many functions are embedded into the hardware which makes the development and maintenance of such honeypots more expensive. Software honeypots are cheaper to develop and maintain. Virtualization technology is also often used as a means to simulate hardware functions [16].

4) Scalability: Scalability refers to the ability of a honeypot or parts thereof to scale. This characteristic becomes particularly important with the emergence of botnets as well as the overall increase in the number of cyberattacks and state actors that aim to attack organizations. Several scalable honeypots exist and they vary from Peer-to-Peer (P2P) functionality to automatic re-deployment mechanisms [17], [18]. Other honeypots developed have focused on handling hundreds of thousands of concurrent connections [5].

B. Honeypot Detection

Honeypot detection lies in an attacker's ability to detect and find out about the deceptive nature of the honeypot. Often this relies on the limited ability of a honeypot to align with an attacker's mental model based on the attacker's expectation of what a realistic environment should look like. The constraints are often economical, which involves hardware as well as development costs. An early example of this is the Honeypot Hunter that was deployed to uncover mail server honeypots [19]. The software utilized its own fake mail server and attempted to proxy back to it through the honeypot's mail server. The action was enough to uncover the honeypot because most honeypot mail servers at the time offered a low overall interaction. There are also additional design constraints for honeypots that attackers are aware of. There are legal constraints that prevent realistic emulations [8]. Such constraints are due to the nature of defenders having to abide by the law and as such attacker's knowingly look for signs of such legal constraints (e.g., a honeypot bot that is part of a botnet cannot legally perform denial of service attacks to systems).

III. Avoiding Honeypot Detection

Given the challenges involved in successfully deploying a non-detectable honeypot, it is important to not only understand recently proposed approaches that have been shown to be successful but also how the characteristics of a honeypot may influence the success of applying these approaches.

A. Proposed Approaches for Avoiding Honeypot Detection

We describe several approaches for honeypots that have been proven to be effective against attacker detection and highlight how they succeed in doing so.

1) Automatic Honeypot Redeployment: Low interaction honeypots are more expensive to develop with built in anti-detection because their scope and functionality are limited in contrast to high interaction honeypots. Upon discovery by an attacker, modifying an existing honeypot configuration may be too costly. Consequently, one approach for avoiding honeypot detection involves automated re-deployment of the honeypot that in turn reduces the need for anti-detection honeypot configurations. One study that has explored the development and implementation of such a mechanism automatically re-deploys a honeynet when it identifies that an attacker has detected it [20]. Inbound and outbound Internet Control Message Protocol (ICMP) packets are monitored and any drop in the number of ICMP packets below a pre-determined threshold indicates that the attacker has disconnected (indicative of the honeypot being discovered). The honeypot is then automatically redeployed with an altered configuration with the intent to trap the same attacker. The method reduces setup overheads and development of anti-detection mechanisms for honeypots.

2) Honeypot Delay Reduction: A substantial detection risk for honeypots stems from the fact that arbitrary delays are introduced in processes that a honeypot mimics (e.g., authentication via SSH that is delayed due to further logging or log forwarding required by a honeypot) that would otherwise not have such delays. These delays enable attackers to detect the honeypot. One method for delay reduction that directly identified the benefits of minimizing delays introduced by an active honeypot bot designed to connect back to a botnet is described in [18]. This method is used to avoid honeypot detection in P2P botnets and it combines the use of IP address spoofing and fake TCP three-way handshake to break the authentication procedure used to allow infected hosts to join a botnet. The authors focused on the Advanced Two-Stage Reconnaissance Worm (ATSRW). The worm uses an authentication procedure before allowing infected hosts to join the botnet. However, the authors of the active honeypot bot discovered that their authentication method that mimicked that of the original botnet was detectable by attackers. The authors of [18] spoofed the authentication by finding an already infected vulnerable peer and infecting it in order to obtain the authentication that was used to authenticate back to the botnet in order to join. The process of the bot learning the authentication procedure introduced a time delay and as a result it revealed that a honeypot was attempting to connect to the botnet. By reducing the time delay to match more closely that of a real infected host, the risk of detecting the honeypot was lowered.

Delay reduction as an optimization for honeypots has also demonstrated its effectiveness in avoiding honeypot's detection when applied to Honeyd (a readily available open-source honeypot) [21]. Honeyd is a virtual (software) honeypot that emulates the protocol stack (such as the TCP/IP stack) so that attackers are convinced they are attacking a real vulnerable system. In Honeyd, an attacker may deduce that the system is a honeypot by measuring the end-to-end latency. Since Honeyd operates on a virtual network, the link latency can be used to detect the presence of the honeypot. Specifically, end-to-end latency in Honeyd's design is a multiple of 10 milliseconds. By comparing latencies between a physical (real) network and a virtual (honeypot) network we can define a threshold in which the two latencies are different. Using that threshold we can detect honeypot networks that use Honeyd. Attackers can use measurements of round trip times (e.g., by using ICMP, TCP, or UDP echo-reply) to detect the presence of honeypots. The study found that camouflaging Honeyd by modifying it to have a lower link latency was effective in avoiding the honeypot system being detected by attackers. The threshold selected was close to the physical network's link latency. The study concluded that although the method was successful with the Honeyd honeypot system, it can also be broadly applied to other virtual honeypot systems.

3) Honeypot Process Transparency: Another side effect of the way honeypots operate is the lack of transparency that leads to revealing their deceptive design. An example of this can be observed in hybrid honeypot systems. In these types of honeypots, both low and high interaction honeypots are used with a frontend honeypot forwarding connections to a backend honeypot (similar to the way load balancing web servers work). Current TCP connection handover mechanisms in hybrid honeypot systems can be easily detected by attackers [5]. As such, one study proposed a transparent TCP connection handover mechanism that uses different ports of an OpenFlow-based switch to isolate honeypots while TCP connection parameters (SEQ, ACK numbers) remain the same [5]. The hybrid honeypots require network traffic to be re-directed from the frontends to the backends and this traffic redirection is often not transparent enough to evade an attacker's detection. The study's alternative mechanism consists of three phases. Phase one is initiated by an attacker sending a TCP connection request to the target honeypot. The honeypot controller forwards the request to the frontend which performs the TCP three-way handshake to establish a connection with the attacker. Phase two involves transferring the TCP session from the frontend to the backend using a TCP replaying approach. This approach replays the three-way handshake using the saved attacker's SYN packet. SEQ and ACK numbers are then synchronized leading to the third phase in which the packets are exchanged directly between the attacker and the backend. The study concluded that this approach makes the hybrid honeypot much stealthier at the expense of reduced performance . In general, honeypots need to ensure that operations that are being emulated need to maintain transparency. In other words, they need to hide any modified sequence of events that is not realistic.

4) Dedicated Hardware: Although expensive, the use of dedicated hardware for honeypots can help minimize their detection (e.g., by reducing arbitrary software delays). Additionally, they also provide an additional layer of security against honeypot compromise. These benefits were demonstrated in a study which showed a more efficient honeypot architecture for malware sample collection. This architecture implemented honeypots using dedicated hardware instead of a general-purpose processor and it also made use of a high-speed implementation of the IP stack [22]. Additionally, the approach used a specialized stateless TCP hardware, which is highly tuned to the application domain. The stateless TCP hardware can manage hundreds of thousands of simultaneous connections thereby enabling the system to support large honeynets. Further, the dedicated hardware used by the honeypot increased the speed of execution of TCP operations, which in turn may reduce the likelihood of a honeypot being detected by an attacker. An additional benefit of implementing honeypot operations in hardware instead of software using general-purpose processors is that it also reduces the risk of the honeypot software being compromised and used for attacks [22].

5) Dynamic Intelligence on Honeypots: The advent of dynamic approaches to honeypot development as well as novel techniques in machine learning and artificial intelligence provide an opportunity for more diverse honeypots that alter their behavior depending on the actions of an attacker. The result is a more adaptable honeypot that is more difficult to detect. In general, dynamic honeypots have a behavior that is not fixed but changes based on some condition and adapts to the current environment [23], [24]. In this case, we do not need to perform configuration or re-configuration. An example of such a system is a high interaction honeypot that uses reinforced learning in order to dynamically change its behavior based on the interaction that it has with an attacker [25]. The honeypot can strategically block program execution, alter program names in order to lure attackers and can deceive attackers with the intent of having them reveal their ethnic background, which is particularly important for research honeypots.

B. Classification Summary of Honeypot Characteristics and Approaches Avoiding Honeypot Detection

The various methods we have described above provide an insight into the diverse array of options that can increase the difficulty of an attacker to detect the deceptive nature of a honeypot. However, many of these approaches may or may not be applicable depending on the honeypot's characteristics. Table I presents a summary of approaches and honeypot characteristics that can work together in order to help honeypot designers better assess their honeypot development and honeypot uses.

Automatic re-deployment can be used for both research and production objectives although it focuses mainly for low interaction honeypots since altering large parts of configuration for high interaction honeypots is expensive. The approach applies to software-based honeypots because hardware based honeypot configuration is too restrictive to yield significant benefits in avoiding detection when altered. The automatic re-deployment method for software-based honeypots is also quite scalable.

By reducing delays in a honeypot system the strategy is effective in reducing the number of hints to attackers for both research and production honeypots. Reduced delays benefit honeypots across all levels of honeypot interaction including hybrid honeypots (involving low and high interactions). The cost of scalability also remains low because many of the parts of the honeypot that can be optimized for avoiding detection are software-based.

Assuring transparency for modified operations (e.g., sand-boxed program execution or TCP connection handover) is important for research honeypots where an attacker is expected to leak more information and have higher level interactions with the honeypot. Since the primary components are software based, they tend to be more scalable as demonstrated by one of the recent studies [22].

Use of dedicated hardware is the most expensive choice in terms of scalability cost but has also been demonstrated to be highly effective for both low and high interaction honeypots. This approach is most suitable when the honeypot interaction's duration is expected to be rather short such as in production honeypots.

Finally, a major benefit for research high interaction honeypots can be achieved through the use of dynamic intelligence in honeypots. The approach is software based and therefore the computational overhead requirements may result in high overheads when scaling up especially if artificial intelligence is involved (that may include long training times).

IV. The Future of Honeypots

Over the past decade several types of honeypots [6], [24], [23], [16] have been proposed with a long history of development . Krawetz [19] predicted the cat and mouse game that has been prevalent in the development of these honeypots. Although honeypot software has not become obsolete, it appears that it has not made the leap that other cybersecurity tools (e.g., Intrusion Detection System (IDS)) have undergone. The majority of honeypots available, although open-source and free, are still monolithic in their architecture with odd development cycles and maintenance. While they are available to security professionals, they are known and highly predictable to attackers with a limited ability to adapt. In fact, more than 50 honeypots that are included in a recent review [6] are falling behind in ensuring detection avoidance from attackers (often having a known vulnerability that signals their deceptive nature). Additionally, many of these tools have terminated their development cycle several years ago. As such, we propose that the new generation of honeypots needs to consider all these shortcomings by embracing a more *adaptive* design, based on more *agile* software engineering paradigms (e.g., microservice) and ensure some *rigidness* against the advent of more advanced detection techniques. We argue that by leveraging AI or machine learning (including adversarial machine learning) [26] we can make the next generation of honeypots more scalable, robust, cost-effective in avoiding their detection from cyber attackers.

By incorporating adaptive techniques in a honeypot's core functionality we can improve "non-discernible" pseudo-randomness in a honeypot's functions and ensure that the honeypot adapts to its network environment. For example, a honeypot software that is placed within a financial network can "learn" based on a training set what files and services typically run in a banking environment and mimic their configuration as close as possible to real-world scenarios. The ultimate goal of this adaptability characteristic is to ensure the persistence of an interaction between an attacker and a honeypot.

Honeypots need to also adopt a modular design by embracing polilithic [27], [28] architecture paradigms. This transformation can ensure that module updates are automatically transferred to the honeypots that use them. For example, a database honeypot can make use of natural language processing and build fake databases on the fly based on the input of an attacker. The decision about making these fake databases and tables will need be structured and can be offered in a modular manner or via an Application Programming Interface (API) (e.g., RESTful API).

Finally, honeypots need not underestimate that machine learning and adversarial machine learning techniques can make static honeypots that do not adapt obsolete. Even those honeypots that support dynamic intelligence (or based on some AI technique) are in danger of becoming detectable unless they retain their training knowledge sets secret and update or alter them frequently.

V. Conclusion

In this work, we have identified several approaches that can be used to ensure detection avoidance for honeypots. We have reviewed and discussed recently proposed successful approaches which should be a first step forward to ensure that honeypots are less detectable by attackers. However, a leap forward is long overdue for honeypot developers to design better deceptive software. New emerging technologies for achieving this goal are already available but a more holistic approach is needed to ensure the next generation of honeypots. Given that honeypots are mainly intelligence gathering devices, their designs and implementations must evolve with

TABLE I
HONEYPOT CHARACTERISTICS AND APPROACHES FOR AVOIDING DETECTION

Detection Avoidance Techniques Approach	**Honeypot Characteristics**			
	Primary Objective	***Interaction Level***	***Implementation***	***Scalability Cost***
Automatic Redeployment	Research or Production	Low Interaction	Software	Low
Delay Reduction	Research or Production	Low & High Interaction	Software	Low
Transparency	Research	High Interaction	Software	Low
Dedicated Hardware	Production	Low & High Interaction	Hardware	High
Dynamic Intelligence	Research	High Interaction	Software	High

the fast changing landscape of cybersecurity threats in the 21st century.

REFERENCES

[1] R. McGrew, "Experiences with Honeypot Systems: Development, Deployment, and Analysis," in *Proceedings of the 39th Annual Hawaii International Conference on System Sciences (HICSS'06)*, vol. 9, 2006, pp. 220a–220a.

[2] C. Stoll, *The Cuckoo's Egg: Tracking a Spy through a Maze of Computer Espionage*. Doubleday, 1989.

[3] C. Hubbard, T. Baker, and M. Tsikerdekis, "Public Entity Network Security Monitoring by Student Security Analysts," in *22nd Colloquium for Information Systems Security Education (CISSE)*, New Orleans, LA, 2018.

[4] L. Spitzner, "The Honeynet Project: trapping the hackers," *IEEE Security & Privacy*, vol. 99, no. 2, pp. 15–23, 2003.

[5] W. Fan and D. Fernandez, "A novel SDN based stealthy TCP connection handover mechanism for hybrid honeypot systems," in *2017 IEEE Conference on Network Softwarization (NetSoft)*, 2017, pp. 1–9.

[6] M. Nawrocki, M. Wählisch, T. C. Schmidt, C. Keil, and J. Schönfelder, "A Survey on Honeypot Software and Data Analysis," Tech. Rep., aug 2016. [Online]. Available: http://arxiv.org/abs/1608.06249

[7] B. Endicott-Popovsky, J. Narvaez, C. Seifert, D. A. Frincke, L. R. O'Neil, and C. Aval, "Use of Deception to Improve Client Honeypot Detection of Drive-by-Download Attacks," in *Foundations of Augmented Cognition. Neuroergonomics and Operational Neuroscience*, D. D. Schmorrow, I. V. Estabrooke, and M. Grootjen, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 138–147.

[8] C. C. Zou and R. Cunningham, "Honeypot-Aware Advanced Botnet Construction and Maintenance," in *International Conference on Dependable Systems and Networks (DSN'06)*, 2006, pp. 199–208.

[9] A. Brown and T. Andel, "What's in your Honeypot?" in *11th International Conference on Cyber Warfare and Security: ICCWS2016*, Boston, MA, USA, 2016, pp. 355–362.

[10] C. Herley and D. Florêncio, "Protecting Financial Institutions from Brute-Force Attacks BT - Proceedings of The Ifip Tc 11 23rd International Information Security Conference," S. Jajodia, P. Samarati, and S. Cimato, Eds. Boston, MA: Springer US, 2008, pp. 681–685.

[11] P. Simoes, T. Cruz, J. Proença, and E. Monteiro, *On the use of Honeypots for Detecting Cyber Attacks on Industrial Control Networks*, jul 2013.

[12] Y. B. Mustapha, H. Débar, and G. Jacob, "Limitation of Honeypot/Honeynet Databases to Enhance Alert Correlation," in *Computer Network Security*, I. Kotenko and V. Skormin, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 203–217.

[13] C. Sanders and J. Smith, *Applied Network Security Monitoring*. Elsevier, 2014.

[14] A. Mairh, D. Barik, K. Verma, and D. Jena, "Honeypot in Network Security: A Survey," in *Proceedings of the 2011 International Conference on Communication, Computing &#38; Security*, ser. ICCCS '11. New York, NY, USA: ACM, 2011, pp. 600–605. [Online]. Available: http://doi.acm.org/10.1145/1947940.1948065

[15] L. Spitzner, *Honeypots: Tracking Hackers*. Boston, MA, USA: Addison Wesley, 2002.

[16] F. H. Abbasi and R. J. Harris, "Experiences with a Generation III virtual Honeynet," in *2009 Australasian Telecommunication Networks and Applications Conference (ATNAC)*, 2009, pp. 1–6.

[17] W. Fan, Z. Du, D. Fernandez, and V. A. Villagra, "Enabling an Anatomic View to Investigate Honeypot Systems: A Survey," *IEEE Systems Journal*, pp. 1–14, 2017.

[18] M. M. Al-Hakbani and M. H. Dahshan, "Avoiding honeypot detection in peer-to-peer botnets," in *2015 IEEE International Conference on Engineering and Technology (ICETECH)*, 2015, pp. 1–7.

[19] N. Krawetz, "Anti-honeypot technology," *IEEE Security & Privacy*, vol. 2, no. 1, pp. 76–79, 2004.

[20] H. Wang and Q. Chen, "Dynamic Deploying Distributed Low-interaction Honeynet," *Journal of Computers*, vol. 7, no. 3, mar 2012.

[21] X. Fu, W. Yu, D. Cheng, X. Tan, K. Streff, and S. Graham, "On Recognizing Virtual Honeypots and Countermeasures," in *Proceedings of the 2Nd IEEE International Symposium on Dependable, Autonomic and Secure Computing*, ser. DASC '06. Washington, DC, USA: IEEE Computer Society, 2006, pp. 211–218. [Online]. Available: https://doi.org/10.1109/DASC.2006.36

[22] S. Mühlbach and A. Koch, "An FPGA-based scalable platform for high-speed malware collection in large IP networks," in *2010 International Conference on Field-Programmable Technology*, 2010, pp. 474–478.

[23] W. Zanoramy Ansiry Zakaria and M. Laiha Mat Kiah, "A review of dynamic and intelligent honeypots," p. 1, 2013.

[24] W. Z. A. Zakaria and M. L. M. Kiah, "A review on artificial intelligence techniques for developing intelligent honeypot," in *2012 8th International Conference on Computing Technology and Information Management (NCM and ICNIT)*, vol. 2, 2012, pp. 696–701.

[25] G. Wagener, R. State, T. Engel, and A. Dulaunoy, "Adaptive and self-configurable honeypots," in *12th IFIP/IEEE International Symposium on Integrated Network Management (IM 2011) and Workshops*, 2011, pp. 345–352.

[26] L. Huang, A. D. Joseph, B. Nelson, B. I. P. Rubinstein, and J. D. Tygar, "Adversarial Machine Learning," in *Proceedings of the 4th ACM Workshop on Security and Artificial Intelligence*, ser. AISec '11. New York, NY, USA: ACM, 2011, pp. 43–58. [Online]. Available: http://doi.acm.org/10.1145/2046684.2046692

[27] G. Kecskemeti, A. C. Marosi, and A. Kertesz, "The ENTICE approach to decompose monolithic services into microservices," in *2016 International Conference on High Performance Computing & Simulation (HPCS)*, 2016, pp. 591–596.

[28] M. Villamizar, O. Garcés, H. Castro, M. Verano, L. Salamanca, R. Casallas, and S. Gil, "Evaluating the monolithic and the microservice architecture pattern to deploy web applications in the cloud," in *2015 10th Computing Colombian Conference (10CCC)*, 2015, pp. 583–590.